Shelled! A Postmortem – By Gary Preston

“Shelled! is an artillery combat game in full 3D that allows up to 8 on-line players to blow each other to bits! Fire a variety of earth-moving explosive turtle shells including nukes, spreads, and more! With classic Scorched Earth inspired game play” - www.shelledgame.com

RedThumbGames began development of Shelled! in January 2005, I joined the team roughly four months in with Shelled! not only being my first TGE based project but also the first real game I'd worked on. The prospect of getting my teeth into a real project was both exciting and at the same time very daunting.

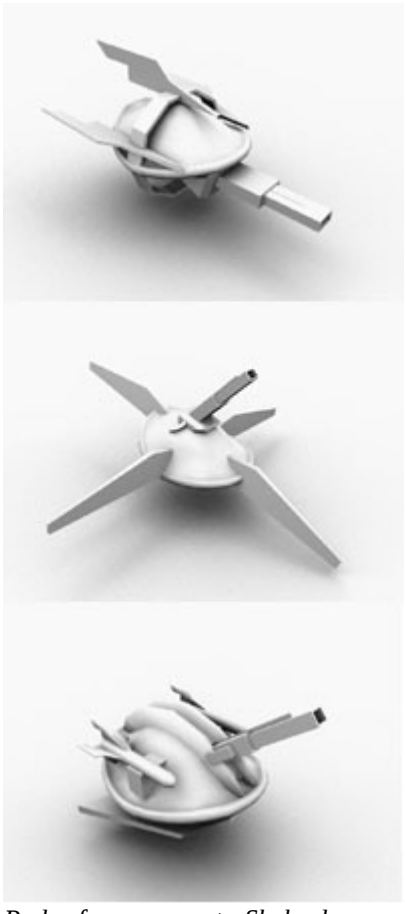
THE TECH

Shelled! was developed using the Torque Game Engine (TGE). For those not familiar with TGE, it's a C++ based engine (some Asm) along with a custom scripting language known simply as "Torque Script".

Torque Script takes a little getting used to, not in terms of its language which is very C like and easy to pickup, but its tight integration with the engine. This integration turns out to also be one of its strengths, if your game were a house, the engine would be the bricks and Torque Script the mortar that holds everything together, the vast majority of our game play logic was script based.

The script code runs several times slower than C++ code despite being compiled to byte code on first use, thus first reactions would be to bypass it and code everything in the engine. This would be a mistake, the productivity gain scripting brings should not be underestimated. It's quite surprising just how much you can script without any noticeable performance impact, besides, it's always possible to move the more demanding script code into the engine during an optimization phase.

That isn't to say Torque Script is perfect, I'm sure most people that have worked with it could come up with a list of extra features they'd like to see, then again you could say that about any engine/development environment.



Pods of war concept - Shelved

A few months after joining the Shelled! team I was made aware of Torsion, an IDE created for editing and debugging Torque Script by SickHead Games. Without doubt productivity increased considerably. For debugging ease alone, I'd strongly recommend that anyone working with Torque Script evaluates Torsion.

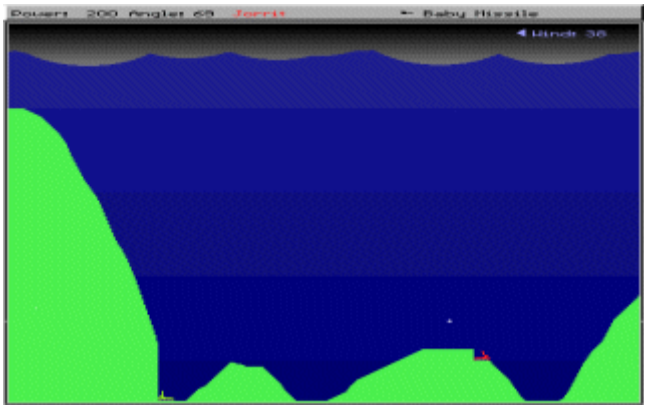
It really goes without saying how critical source control is for any project especially those done remotely. We used CVS for most of the project switching to SVN nearer the end.

During the last few months of the project we also setup Mantis for logging all reported bugs from the latest beta test. In hindsight we should have had mantis up and running from the start. The effort in co-ordinating who was fixing which bug and whether a bug had been fixed or not was greatly reduced. It did take a little more time up front to get bugs logged, but the later time savings were more than worth it, even more so if your project team is split across several timezones.

WHAT WENT WRONG

OUT FEATURE THE COMPETITION

From the off Shelled! had a clear goal, to recreate the fun factor of the classic Scorched Earth game in 3D with simple controls and game play. Early on we lost sight of that goal.



Scorched Earth - Inspiration for Shelled!

Several months into development the game had it all, flying tanks, deformable terrain, loads of weapons, various planets sporting different gravity levels, tank power ups and a wind system that would occasionally change direction and affect shell trajectories, to mention but a few. What we didn't have, was accessible game play.

The game was too cumbersome, it had too many options, the number of clicks needed to go from loading the game to actually firing your first in shell in anger was bordering on the insane. Selecting a level required setting min/max bot skill level, min/max wind speeds, fixed or varied wind direction, planet gravity, terrain type, allowed weapons, points/money for kills and a host of other options that whilst some may find useful, for the majority it was just over whelming.

Looking back it was hardly surprising the early versions of the game met with an amount of criticism during the first public viewing. In short order the game was ripped apart and just about every major feature ripped out and game controls overhauled. The user interface was stripped and reduced to a fixed set of simple options. Now the user could get from main menu into an actual game in just a few clicks.

LEARNING CURVE

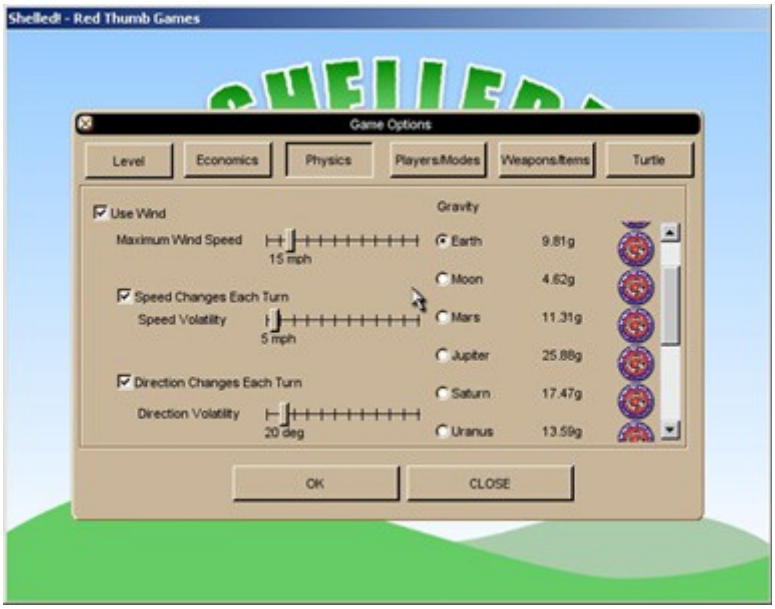
TGE is a sizable chunk of code, the learning curve of which is fairly steep. This steep learning curve was compounded by my initial mindset of wanting to understand how everything worked before making changes. This is probably due to my Information Systems background, where significant time was often spent understanding new systems before any changes were made.

With an engine like TGE this wasn't feasible. Not only due to the sheer amount of code, but also because understanding much of the code requires that you already have a grasp of several concepts unique to the game domain.

Once I'd shaken off the need to understand everything and instead focused on only code in the immediate area of concern, I found my overall knowledge of the engine increased at a greater pace.

CODE WHAT YOU NEED, NOT WHAT YOU THINK YOU MIGHT

A few times whilst adding support for a specific feature I would catch myself over analyzing what was needed and looking for a "perfect" solution that not only supported the current task but also allowed easy expansion in the future. On the surface this isn't a bad thing to do and in moderation can be very useful. However, when you try to be too flexible or always look for the "best" way to do something, you can end up spending hours/days longer adding a feature that will also support XYZ only to find you never needed X let alone Y or Z. All that extra time making it flexible was for naught and the extra fluff may have reduced the readability of the code. Even worse, you may find the feature gets cut a few months down the line.



Custom Options - Cut

Coding only “what you need” became an important lesson for me and fortunately one I learned very early on in the project. Should a feature later be required, then, at that time and not before, re-factor and add the new feature. In pretty much every case the flexibility I thought we might need later was never the case and in most cases features were later scaled back or dropped.

USABILITY TESTING

We left it quite late in the project before opening the game to a wider audience of testers. By which time a number of features had been fully implemented and final artwork put in place. Only to then discover the intended target audience for the game were overwhelmed by the amount of options and general control scheme.

High quality artwork was thrown out and the theme changed to a more cartoon style, but the changes didn't end there. Feature after feature found itself mercilessly cut, some of which had taken considerable time to implement over the previous months.

This kind of design change can be minimized by prototyping up front (lesson learned) however even with prototyping, games are creative works and will under go changes throughout the development life cycle. Features may be dropped due to feedback from testers or cut due to budget/time constraints.

Such a major change in direction so far into development was a hard to do, but once the dust settled and newer “cartoon” style artwork was in place, Josh's decision was vindicated and the game really started to finally come together. Had we not followed the "CODE WHAT YOU NEED" rule, this cutting of features would have been all the more painful.

In hindsight we should have prototyped the main features of the game and done a wider form of testing with people from the target market rather than relying on our own opinions.

CROSS PLATFORM SUPPORT

TGE is a cross platform engine that works on Windows, MacOS and various flavors of Linux. In the early days of development, Shelled! compiled and ran on all three. However it wasn't long before development focused entirely on Windows with the intention of bringing Mac/Linux up to date at a later time. As development neared an end, we had a game that no longer compiled let alone ran on the Mac nor Linux. Bringing the game up to date on those platforms probably wouldn't have taken too long, but it was time we no longer had.

In hindsight we would have been better off keeping Mac and Linux builds up to date either after each feature change or at regular milestones. The time it would take to resolve issues just after a feature change whilst the changes were still fresh would probably have been lower than the time it was going to take to resolve everything a year or two later at the projects end.

In addition testing of new features could have been completed on each OS throughout development, by the end of the project even if we ported the game to the Mac and Linux we'd have been left with a sizable amount of feature testing to do. Any major platform bugs would have then caused a significant delay, where as the same bugs found during development could have been resolved by bringing in a Mac/Linux programmer to work in parallel.

Original Armory and tank power-ups
Later reduced to 9 shells

As a result, Shelled! has only been released on the Windows Platform.

Early Sci-Fi Themed Build - Cut

AI

I guess this doesn't really belong in the "what went wrong" category, more in the, "if we had more time" category.

With the main emphasis on network play the AI took a bit of a back seat. This becomes quite evident in that the AI will only ever make use of the default shell type. They're incapable of purchasing and making the decision as to when to use different shell types.

Artificial Intelligence is a fascinating subject with a number of techniques that would have been worth investigating for use in Shelled!, time however was not on our side.

Mech Pilot for game victory celebrations - Cut

NETWORKING

Shelled! is mainly a multi player game and unfortunately one key problem is the requirement of users to have ports open before their "hosted" game will appear on the master server. Although many AAA games have this problem, they also have such a large pool of players that its almost a given that a sufficient proportion will know how to forward a few ports to let them run their own servers.

Unless your game is aimed at hardcore gamers we cannot assume this level of technical knowledge. There will of course be exceptions, but can we count on that? One potential problem being players download the game, click on-line play and see no servers, so they click host and sit playing the AI for a while then decide nobody else is going to join them and quit. It could be that a number of other players were doing the exact same thing and yet none of them were aware of each other.

This could be addressed through running a few dedicated servers but ideally we should removing the need for the end user to worry about port forwarding at all. With UDP packets and the co-operation of a master server, it's possible to traverse firewalls and NAT'd connections. Several methods are available but we didn't have the time needed to investigate and implement any. I believe TNL has an implementation (used by ZAP!?) a feature that would make a worth while addition to TGE for future projects.

WHAT WENT RIGHT

COMMUNICATION

Communication on remote projects is problematic at the best of times, but add to that an 8 hour timezone difference and incompatible working hours making real-time communication impossible within any kind of reasonable hour and the potential for problems increases.

The alternative, Email and IRC. This wasn't as bad as it sounds, we had a paper trail of feature requests and bugs fixes, in addition feature requests were clear and concise which greatly reduced the chance of differing interpretations.

FAKING IT

The initial implementation of the Tanks and Jets attempted to model forces such as jet thrust, wind, gravity and surface friction amongst others. It quickly became apparent that implementing "realistic" physics (as realistic as flying, nuke wielding turtles can be anyway ;) was simply getting in the way of the game play.

Flying Tank - Final Build

A few people have commented that they liked the physics in Shelled! and yet under the hood they're very unrealistic. Aside from simulations, accurate physics are not that important so long as the implementation "feels" good it shouldn't matter whether you have a cool rigid body simulation that takes into account thrust/mass/gravity or simply a set of fudged numbers.

One example of this is the initiation of flight, the original jet thrust gradually accelerates the tank upwards. However, in the final build we went instead for an instantaneous velocity followed by a gradual acceleration that tapered off quickly. This was done to allow tanks to exit craters quickly yet limit the final height they could achieve.

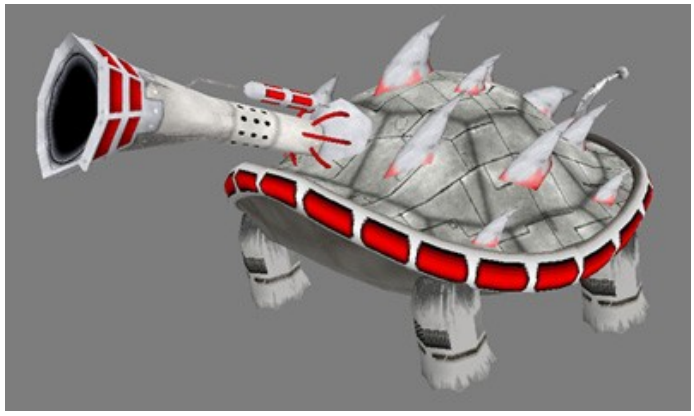
Another area where game play came before reality was the way tanks conform, or more specifically do not conform to the terrain. This was done for two reasons. First, having a fixed level platform made the AI trajectory calculations easier. Second, and of greater importance. The concept of "trajectory" based game play, getting players to aim above a target rather than directly at it, was already a hard sell for our main audience of players. We'd have only complicated matters if tanks could come to a rest angled against a cliff face and the player suddenly found themselves having to move left/right rather than up/down aim higher or lower.

We didn't want to be "too" apparent that we'd taken this approach and landing a tank on the top of a spire made it obvious that we'd faked it. Again we fudged it, instead of tumbling off the spire we applied a small force in the direction of the largest overhang to push the tank clear. Accurate, no, but it did the trick.

COMMUNITY RESOURCES

These were a mixed bag. On the one hand, we saved quite a substantial amount of time and effort by leveraging the work of the community, however, we also lost time tracking down and fixing various bugs in the resources, many of which went unnoticed for some time. In several cases the resources were later removed in favor of custom code. Although we still saved time by initially using resources to quickly test a concept and as a basis for more customized code.

Notable exceptions that remained in the final build were resources such as Ben Garney's Commander Map and Stephan Zepp's terrain deformation code both of which saved us a great deal of research and implementation time. Neither resource remained untouched however, we made substantial changes to the terrain deformation code to tailor it to our needs. Such as ensuring any objects in the vicinity had their cached collision sets invalidated to ensure they fell into craters. We also added a crater impact texture and abused the "updateGridMaterials" code to play nicely with tiled terrain.



Enemy Mech Tank - Cut

The Garage Games community as a whole are a great asset to the Torque Game Engine. Both in terms of supporting each other through forums and IRC as well as taking the time to produce resources, which, when carefully selected can greatly aid development.

JOURNALING

Once again the choice of engine paid dividends. TGE has the ability to record and play back demos of your game to help remind other gamers of how you gracefully nuked them out of existence, however, a less commonly used feature is journaling.

Journaling records all the input the game receives whilst running allowing a game session to be replayed again at a later time, but unlike a demo, you can replay the session from within your favorite debugger. If beta testers are having strange crash bugs that you can't reproduce, make them play the game with journaling enabled and send you the resulting file. Fire up visual studio and replay the journal in debug mode, go make a coffee and when you come back the debugger should have broken out just before the event that causes the crash to occur.

Journaling helped track down several major bugs within Shelled! Although we never needed beta testers to send in a journal file, the option was there and would also help remove any ambiguities that an emailed bug report could contain.

NETWORKING

Using Torque allowed us to make full use of its great networking support. There were a few teething problems but these were more down to inexperience with the networking side of the engine. We also spent a little time testing Shelled under fairly high packet loss and latency's of up to 200-350ms, with only one exception, the game still played well. The exception being the purchase screen which was re-written to use client side predication based on the clients last confirmed Cash amount.

CLOSING THOUGHTS

When things go wrong, you find yourself having to work even harder to solve the problem. That extra effort tends to make those events easier to recall. I believe this is why a lot more appears to have gone wrong with the project than right. In reality the two were probably well balanced, mistakes are just easier to remember.

What I personally took home from the project (aside from a neat Shelled! branded mug) was a deeper understanding of the engine as well as a greater appreciation for what goes into making a complete game, although I still feel like I've only scratched the surface of TGE in many areas. Perhaps most importantly, I've realized just how much I didn't know I didn't know about game making ;)

Did we manage to make a fun game? I think we did. The final decision is of course up to the players, head on over to www.shelledgame.com and give the game a whirl.

Gary Preston
Lead Programmer
gary@figmentgames.com

PROJECT VITALS

Development software: 3D Studio Max, Visual Studio.net 2003, Subversion, Mantis, Torsion
Engine Used: Torque Game Engine
Development Hardware: 2.8GHz HT P4, 1Gig Ram, 256Meg GF6800
Release Platform: Windows

Programmers: 1-2
Artists: 1
Release Date: 14th November 2006

Free Download: http://www.shelledgame.com

VGCore Score: 8.8 / 10
VGCore Review: http://pc.vgcore.com/reviews/445.html
Game Tunnel: http://www.gametunnel.com/gamespace.php?id=306